

# Estruturas de Dados e Algoritmos

## Capítulo 4 Programação Orientada a Objetos

## Programação Orientada a Objetos

- Classes de Objetos
- Construtor e Destruitor
- Ponteiros para Objetos
- Sobrecarga de Operadores
- Membros Estáticos
- Funções e Classes Amigas
- Herança de Classes
- Métodos Virtuais e Classes Abstratas
- Templates de Classes

1<sup>a</sup> Aula

# Programação Orientada a Objetos

- Classes de Objetos

# Programação Orientada a Objetos

- Paradigma de program. dominante desde os anos '90.
- Conceitos importantes: **Objeto** e **Classe de Objetos**
  - **Objeto**: estrutura de dados que inclui **atributos** e **operações**.
    - As operações são designadas por **métodos** em C++.
  - **Classe de Objetos**: define as características comuns de um conjunto de objetos do mesmo tipo, ou seja, objetos que possuem os mesmos atributos e métodos.
    - Declaradas em C++ através da palavra reservada `class`.
    - Permitem a manipulação de dados que podemos fazer com estruturas declaradas através de `struct` e muito mais...

## Classe – Declaração/Definição

```
class nomeClasse {
[tipoVisibilidade_1:]
    membro_11;
    membro_12;
    ...
[tipoVisibilidade_2:]
    membro_21;
    membro_22;
    ...
} [nomeObjectos];
```

- nomeClasse – nome da classe;
- nomeObjectos – lista de nomes de objetos (opcional);
- membro\_ij – **membro**, i.e. atributo ou operação da classe;
- tipoVisibilidade\_i – visibilidade dos membros declarados a seguir.

## Visibilidade dos Membros

- private – apenas acessíveis a outros membros da mesma classe ou a objetos de classes «amigas»
- protected – acessíveis a outros membros da mesma classe, a objetos de classes «amigas», ou a membros de classes derivadas
- public: acessíveis de qualquer ponto do programa onde a classe seja visível
  - Por omissão, se declararmos membros de uma classe antes de especificarmos qualquer tipo de visibilidade, então esses membros são do tipo private

## Declaração da Classe

```
class CRetangulo{
    int x, y;
public:
    void inicializaValores(int,int);
    int area(void);
} Ret;
```

- Declaração do objeto Ret da classe CRetangulo:
  - 4 membros
  - 2 atributos private
  - 2 métodos public

- Ao declarar-se a classe, pode-se também definir completamente os seus métodos
- Recomenda-se que os métodos sejam apenas declarados (apenas os seus protótipos), sendo definidos mais à frente com o operador :: de resolução de espaço de nomes

## Referenciar Membros

```
int area;

Ret.inicializaValores(3, 4);
area = Ret.area();

CRetangulo *pR = &Ret;
pR->inicializaValores(6, 2);

Ret.x = 5; // Erro: x é private!
```

- A referenciação de um objeto é semelhante ao que se faz com struct:
   
nomeObjeto.nomeMembro
- Com um ponteiro:
   
ponteiroObjeto->nomeMembro
   
(\*ponteiroObjeto).nomeMembro
- Dá erro a compilar se violarmos as regras de visibilidade dos membros

## Definição dos Métodos

```
#include <iostream>

class CRetangulo{
    int x, y;
public:
    void inicializaValores (int, int);
    int area (void) { return x*y; } ←
    int perimetro (void) { return 2*x+2*y; } ←
};

void CRetangulo :: inicializaValores (int a, int b)
{ x = a; y = b; }

int main () {
    CRetangulo Ret;
    Ret.inicializaValores(3, 4);           // Ret.x = 3; Ret.y = 4
    std::cout << "Área: " << Ret.area(); // área = 12
}
```

■ O operador `::` define o escopo do atributo ou método, i.e. a que classe diz respeito

Métodos “inline”

## Instanciação de Objetos

- Uma das vantagens da definição de classes de objetos é poder-se instanciar múltiplos objetos com características similares, i.e. com os mesmos atributos e métodos

```
int main () {
    CRetangulo RetA, RetB;
    RetA.inicializaValores (3,4);
    RetB.inicializaValores (5,6);
    cout << "Área RetA: "
        << RetA.area() << endl;
    cout << "Área RetB: "
        << RetB.area() << endl;
}
```

Área RetA: 12  
Área RetB: 30

# Programação Orientada a Objetos

- Classes de Objetos
- Construtor e Destruitor

# Construtor e Destruitor

- O **construtor** de uma classe é chamado automaticamente quando é criado um objeto
  - Usado habitualmente para inicializar os atributos do objeto
- O **destrutor** de uma classe é chamado automaticamente quando é destruído um objeto
  - Usado habitualmente para libertar memória dinâmica associada à estrutura de dados do objeto que está prestes a ser destruído
    - Nem sempre é necessário...

## Construtor e Destruitor

```

class CRetangulo {
    int *pAltura, *pLargura;
public:
    CRetangulo();           // construtor por defeito
    CRetangulo(int, int);   // construtor por enumeração
    ~CRetangulo();          // destrutor
    int area(void) { return ( (*pAltura) * (*pLargura) ); }
};

CRetangulo :: CRetangulo() { //construtor por defeito
    pAltura = new int; pLargura = new int;
    *pAltura = 1; *pLargura = 1; }

CRetangulo :: CRetangulo(int a, int b) { //construtor por enumeração
    pAltura = new int; pLargura = new int;
    *pAltura = a; *pLargura = b; }

CRetangulo :: ~CRetangulo () { // destrutor
    delete pAltura; delete pLargura; }

```

## Sobrecarga do Construtor

```

class CRetangulo {
    int *pAltura, *pLargura;
public:
    CRetangulo();           // construtor por defeito
    CRetangulo(int);        // sobrecarga do construtor por enumeração
    CRetangulo(int, int);   // construtor por enumeração
    ~CRetangulo();          // destrutor

    void inicializaValores(int, int);
};

...
CRetangulo :: CRetangulo(int a) { //sobrecarga construtor por enumeração
    pAltura = new int; pLargura = new int;
    *pAltura = a; *pLargura = a; }

void CRetangulo :: inicializaValores(int a, int b) {
    *pAltura = a; *pLargura = b; }

```

# Construtores Definidos por Defeito

- Construtor por cópia:
  - Permite fazer uma cópia de um objeto, para inicializar um novo objeto através da cópia de um outro, ou para passar por valor um objeto a uma função
- O compilador define por defeito:
  - O construtor por defeito (sem parâmetros), se não implementado explicitamente na classe
  - O construtor por cópia se não implementado explicitamente:
    - Por defeito, faz uma cópia “attribute-wise” de todos os atributos
    - Não desejável quando o objeto usa memória dinâmica!...
  - A sobrecarga do operador = (explicado mais à frente...)

# Construtor por Cópia

```

class CRetangulo{
    int *pAltura, *pLargura;
public:
    ...
    CRetangulo(const CRetangulo &); // construtor cópia
};

CRetangulo :: CRetangulo(const CRetangulo &um) {
    pAltura = new int; pLargura = new int;
    *pAltura = *(um.pAltura); *pLargura = *(um.pLargura);
}

int main() {
    CRetangulo a; // é chamado o construtor por defeito
    CRetangulo b(2, 3); // é chamado o construtor por enumeração
    CRetangulo c(4); // é chamado o construtor que só aceita um parâmetro
    CRetangulo d(b); // é chamado o construtor por cópia

    cout << "Áreas de a, b, c, d:" << endl
        << a.area() << ' ' << b.area() << ' ' << c.area() << ' ' d.area();
}

```

Áreas de a, b, c, d:

1 6 16 6

# Programação Orientada a Objetos

- Classes de Objetos
- Construtor e Destruitor
- Ponteiros para Objetos

# Ponteiros para Objetos

- É possível ter ponteiros para objetos como para outros tipos de dados (ex. int, struct, etc.)

```
int main() {
    CRetangulo a, *b, c(3,2), *d;
    CRetangulo *e = new CRetangulo [2];
    b = new CRetangulo;
    b->inicializaValores(3,4);
    d = &c;
    e[1].inicializaValores(7,8);
    cout << "Área (a): " << a.area() << endl;
    cout << "Área (b): " << b->area() << endl;
    cout << "Área (c): " << c.area() << endl;
    cout << "Área (d): " << d->area() << endl;
    cout << "Área (e[0]): " << e[0].area() << endl;
    cout << "Área (e[1]): " << e[1].area() << endl;
    delete b; delete [e];
}
```

Área (a): 1  
 Área (b): 12  
 Área (c): 6  
 Área (d): 6  
 Área (e[0]): 1  
 Área (e[1]): 56

## Palavra Reservada “this”

- A palavra `this` pode ser utilizada na definição de um método de uma classe (não pode ser usada fora desse contexto)
- Representa um ponteiro para o objeto para o qual o método é chamado

```
class CTeste{
public:
    bool souEU (CTeste& um) {
        if (&um == this) return true;
        else return false; }
};

int main() {
    CTeste a, b, *c = &a;
    if (c->souEU(b)) // FALSO
X cout << "Sou o 'b'.";
    if (c->souEU(a)) // VERDADEIRO
✓ cout << "Sou o 'a'.";
```

## Palavra Reservada “this”

- A palavra `this` permite que um método devolva uma referência

```
class Data {
    int dia, mes, ano;
public:
    Data& initDia(int d);
    Data& initMes(int m);
    Data& initAno(int a);
};

// Método initDia() devolve uma referência para o próprio objeto
Data& Data :: initDia(int d){
    this->dia = d; // podia ser 'dia=d'
    return *this;
}
...
Data dNascim;
dNascim.initDia(13).initMes(5).initAno(1973);
```